SEGMENT_FDM Version 1
Software user manual

ALS 53331 b-en

First issue: 04-2000
This edition: 07-2000

Meaning of terms that may be used in this document / Notice to readers

WARNING

Warning notices are used to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist or may be associated with use of a particular equipment.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

Caution

Caution notices are used where there is a risk of damage to equipment for example.

Note

Notes merely call attention to information that is especially significant to understanding and operating the equipment.

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all systems. ALSTOM assumes no obligation of notice to holders of this document with respect to changes subsequently made.

ALSTOM makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. ALSTOM gives no warranties of merchantability or fitness for purpose shall apply.

In this publication, no mention is made of rights with respect to trademarks or tradenames that may attach to certain words or signs. The absence of such mention, however, in no way implies there is no protection.

Partial reproduction of this document is authorized, but limited to internal use, for information only and for no commercial purpose.

However, such authorization is granted only on the express condition that any partial copy of the document bears a mention of its property, including the copyright statement.

All rights reserved.
© Copyright 2000. ALSTOM (Paris, France)

Index letter	Date	Nature of revision
b	**07-2000**	Modifications throughout the document

Revisions

1. PURPOSE OF MANUAL AND DOCUMENTED VERSION

This document is the user manual for the SEGMENT_FDM Software, version 1. This is a subscriber message segmentation software built with the FULLFIP2 communication component.

This manual describes all the functions offered by this software, and how to implement and use it.

2. CONTENT OF THIS MANUAL

The content of this manual is as follows:

Chapter 1: **General presentation**: description of the Software, functions, presentation with the various interfaces

Chapter 2: **Description of functions**: operating contexts, transfers

Chapter 3: **Primitives**: description of primitives

Chapter 4: **Software Initialisation**: operation of the software

Chapter 5: **Error messages**

Chapter 6: **Installation procedures**

Appendix A: **Example**

Glossary

3. RELATED PUBLICATIONS

The documents mentioned in this manual are specified in the text between brackets and listed in this paragraph:

- Y3-30 A420597-A User requirements
- Y3-30 A418811-A Segmentation software development plan
- ALS 50278 FIP DEVICE MANAGER Software Version 4 User Reference Manual

4. WE WELCOME YOUR COMMENTS AND SUGGESTIONS

ALSTOM strives to produce quality technical documentation. Please take the time to fill in and return the "Reader's Comments" page if you have any remarks or suggestions.

Preface

Reader's comments

ALS 53331 b-en	*SEGMENT_FDM Version 1 Software user manual*

Your main job is:

- ❑ System designer
- ❑ Distributor
- ❑ System integrator
- ❑ Installer
- ❑ Programmer
- ❑ Maintenance
- ❑ Operator
- ❑ Other (specify below)

If you would like a personal reply, please fill in your name and address below:

COMPANY:.....NAME:.....

ADDRESS:.....

.....COUNTRY:.....

Send this form directly to your ALSTOM sales representative or to this address:

ALSTOM Technology
Technical Documentation Department (TDD)
5, avenue Newton - B.P. 215
92142 CLAMART Cedex
France
Fax: +33 (0)1 46 29 12 44

All comments will be considered by qualified personnel.

REMARKS

Reader's comments

Contents

CHAPTER 1 - GENERAL PRESENTATION

1. GENERAL SOFTWARE DESCRIPTION 1-1
2. PRESENTATION OF FUNCTIONS 1-3
3. PRESENTATION OF THE INTERFACE WITH THE USER APPLICATION 1-4
   - 3.1 Initialisation primitives 1-4
   - 3.2 Configuration functions 1-4
   - 3.3 Send messages functions 1-4
   - 3.4 Receive messages functions 1-5
   - 3.5 Activation functions 1-5
   - 3.6 Functions to inform the user 1-5
4. PRESENTATION OF THE INTERFACE WITH FDM_R4 1-6
5. PRESENTATION OF THE INTERFACE WITH THE OPERATING ENVIRONMENT 1-7

CHAPTER 2 - DESCRIPTION OF FUNCTIONS

1. ADMINISTRATION OF OPERATING CONTEXTS 2-1
   - 1.1 Creation of general operating context: MCS instance 2-2
   - 1.2 Inform the user of operating faults 2-3
   - 1.3 Sequencing the actions to be performed 2-4
2. CHANNEL COMMANDS 2-6
   - 2.1 Channel configuration 2-6
3. TRANSFERS 2-8
   - 3.1 Sending a message on a channel 2-9
   - 3.2 Receiving a message on a channel 2-10

CHAPTER 3 - PRIMITIVES

1. DESCRIPTION OF USER PRIMITIVES 3-1
   - 1.1 Procedure mcs_get_software_version 3-1
   - 1.2 Procedure mcs_initialize 3-2
   - 1.3 Procedure mcs_fifo_empty() 3-4
   - 1.4 Procedure mcs_get_errorandwarningstring 3-5
   - 1.5 Procedure mcs_channel_command 3-5
   - 1.6 Procedure mcs_send_message () 3-10
   - 1.7 Procedure mcs_free_received_message() 3-13
   - 1.8 Procedure mcs_cat_msg_received() 3-14
   - 1.9 Primitives called, to be supplied by the user 3-15
2. DESCRIPTION OF OS PRIMITIVES 3-18
   - 2.1 OS primitives of FDM_R4 3-18
   - 2.2 OS primitives for SEGMENT_FDM 3-18

Contents

CHAPTER 4 - SOFTWARE INITIALISATION

1. HOW TO RUN THE SOFTWARE 4-1
2. INITIALISE FDM_R4 4-2
   - 2.1 Use of the Periodic Message system 4-2
   - 2.2 Use of the Aperiodic Message system 4-2
3. RUN AN APPROPRIATE BUS ARBITER PROGRAM 4-3
4. INITIALISE THE SOFTWARE AND CONFIGURE THE "MCS CHANNELS" 4-4

CHAPTER 5 - ERROR MESSAGES

1. CODE _MCS_ERROR_CODE_LIST; 5-1
2. CONNEXION_FAILURE 5-4

CHAPTER 6 – INSTALLATION PROCEDURES

1. PRESENTATION OF DELIVERY 6-1
2. IMPLEMENTATION 6-2
3. COMPILATION PARAMETERS 6-3

APPENDIX A - EXAMPLE

EXAMPLE OF SEGMENT_FDM SOFTWARE MAKEFILE FOR PSOS A-1

GLOSSARY

Chapter 1 General Presentation

1. GENERAL SOFTWARE DESCRIPTION

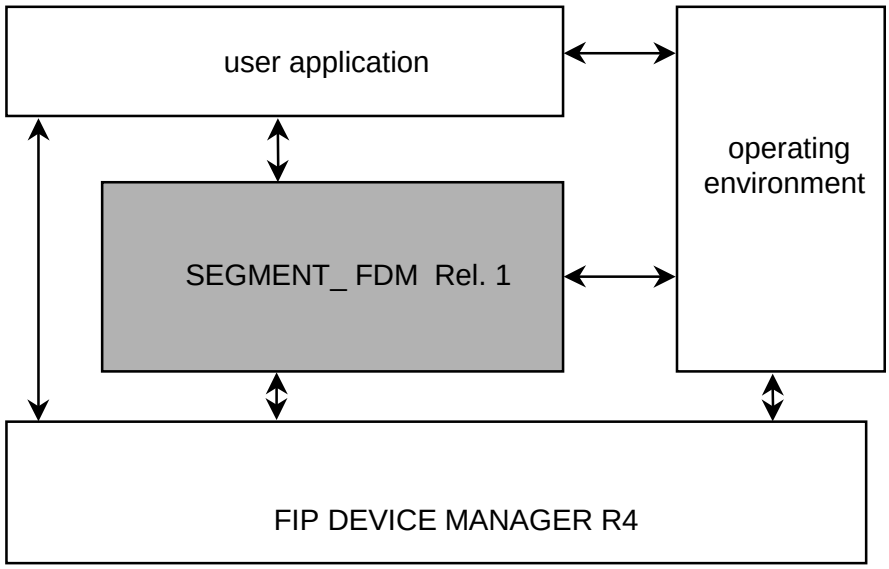
The role of the Software is to enable messages of any size to be transferred on the WorldFIP network.

It is responsible for breaking down the message into a size compatible with that of the frames in the WorldFIP data link layer, and for reassembling them at the receiver.

It is designed to be run on embedded targets, equipped with the FULLFIP2 communication component.

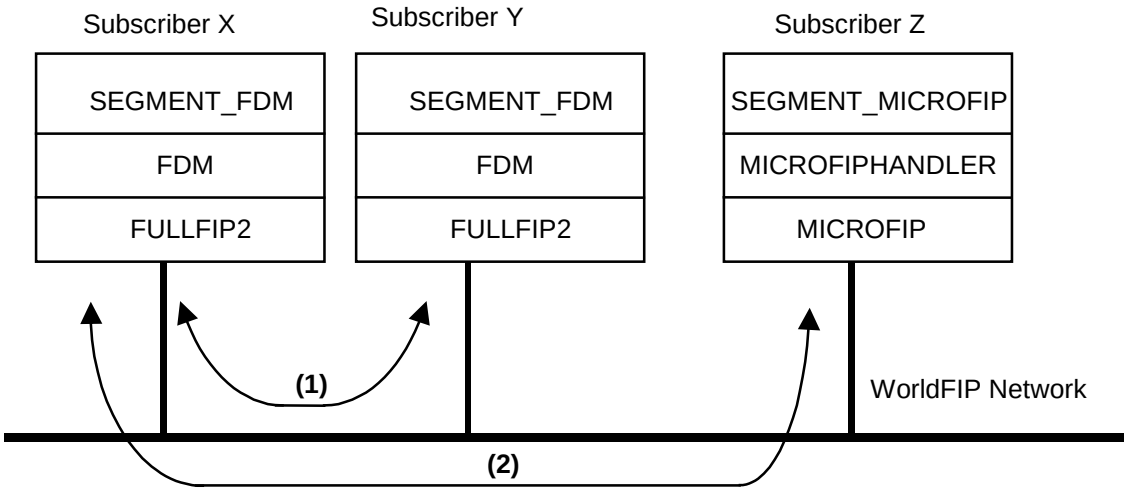
It is delivered in the form of a library of primitives to be linked with the user application and the FIP DEVICE MANAGER R4 software.

It is written in C ANSI and developed regardless of its operating context.



The Software offers an interface for the users of a station wishing to exchange messages with another subscriber.

The messages exchanged are larger than those which can be transmitted in a frame.



The message frames on the WorldFIP network differ according to the type of subscriber connected:

- for a connection to a subscriber on FULLFIP2, the frame is 256 bytes,
- for a connection to a subscriber on MICROFIP, the frame is 122 bytes.

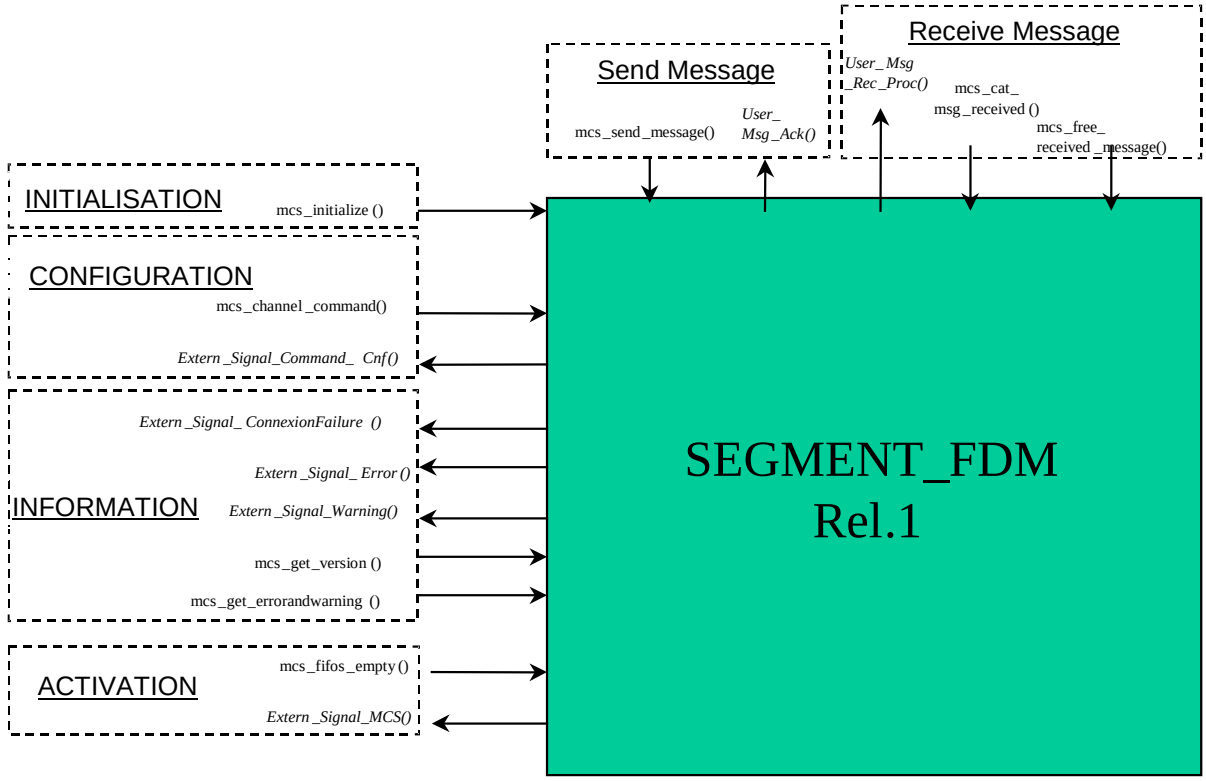
2. PRESENTATION OF FUNCTIONS

The Software offers the following functions:

- system operation:
  - administering operating contexts:
    - initialisation of the general operating context,
    - configuration (creation, stop and delete) of SEGMENT_FDM channels,
  - informing the user of operating faults,
  - sequencing the actions to be performed,
- transfer of data via an A_DATA channel using the MCS protocol:
  - sending a message on a channel,
  - receiving a message on a channel.

3. PRESENTATION OF THE INTERFACE WITH THE USER APPLICATION

This interface offers the user primitives which are associated with the various SOFTWARE functions:



3.1 Initialisation primitives

mcs_initialize	software initialisation (creation of the instance).

3.2 Configuration functions

mcs_channel_command	channel operating command: creation, stop, destruction.
Extern_Signal_Command_cnf	informs the user when a channel command has been acknowledged.

3.3 Send messages functions

mcs_send_message	sends a message on a channel
User_Msg_Ack	informs the user when a message has been acknowledged

3.4 Receive messages functions

User_Msg_Rec_Proc	informs the user when a message is received.
mcs_free_received_message	releases a received message.
mcs_cat_msg_received	concatenation of received message data.

3.5 Activation functions

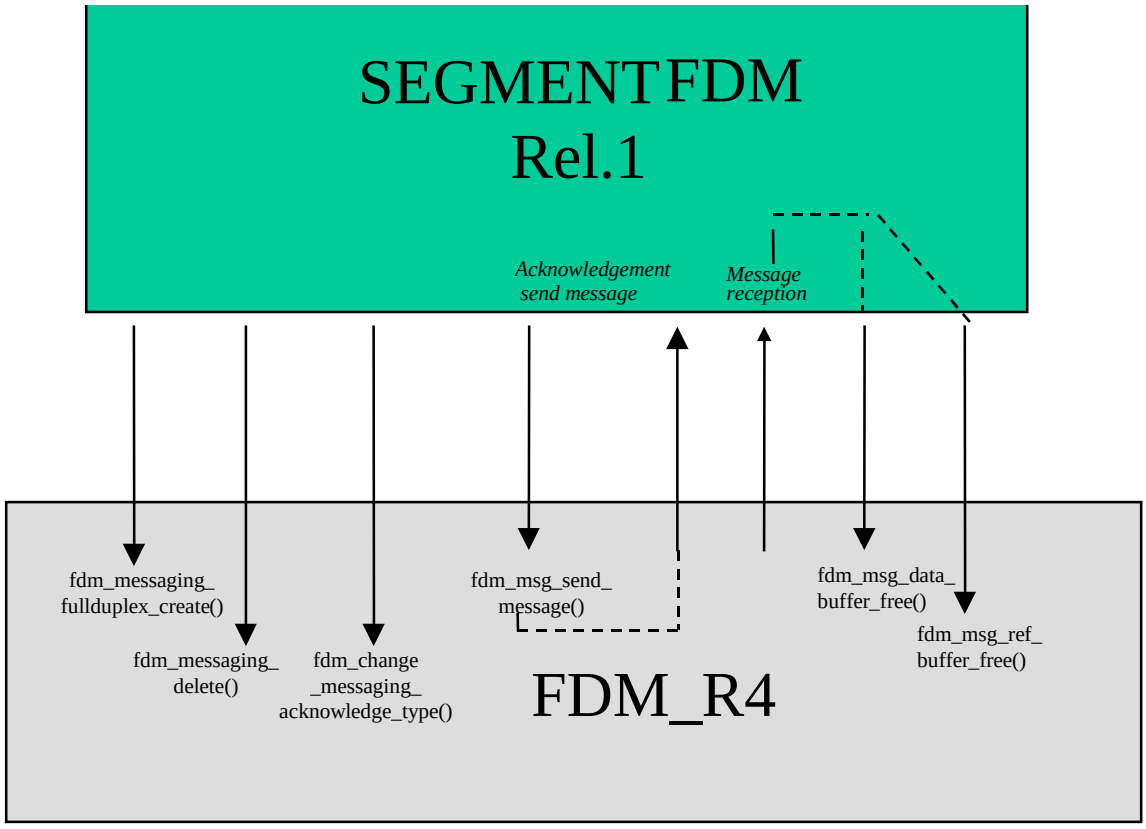
mcs_fifos_empty	activation of Software internal fifos emptying function.
Extern_Signal_MCS	informs the user that he must activate the Software internal fifos emptying task.

3.6 Functions to inform the user

Extern_Signal_Error	informs the user of a serious error
Extern_Signal_Warning	informs the user of a minor error
Extern_Signal_ConnexionFailure	informs the user of a connection failure
mcs_get_version	Software version read request
mcs_get_errorandwarningstring	error or warning code in character string format

4. PRESENTATION OF THE INTERFACE WITH FDM_R4

Internally, the Software calls on the FIP DEVICE MANAGER message services.



When configuring the channels, it also calls on:

fdm_messaging_fullduplex_create: creation of message context for sending and receiving.

fdm_messaging_delete: destruction of message context.

fdm_change_messaging_acknowledge_type: modification of the messaging acknowledgement type

fdm_msg_send_message(): for sending it calls on.

fdm_msg_ref_buffer_free()
and *fdm_msg_ref_data_buffer_free()*: for reception it calls on.

5. PRESENTATION OF THE INTERFACE WITH THE OPERATING ENVIRONMENT

The operating environment interface is the same as that described for FDM_R4:

(see manual ALS 50278, chapter 1, Section 4**: "**Overview of interface with operating environment").

The user must therefore carry out the actions requested for FDM_R4.

The OS-related procedures described by FDM_R4 and used by the Software are as follows:

- **OS_Enter_Region()** **OS_Leave_Region(),**
- **OS_semaphore_take()** **OS_semaphore_release().**

The Software also uses binary semaphore. The user must therefore write the semaphore create and delete procedures:

- **OS_BinarySemaphoreCreate()** and
- **OS_Semaphore Delete().**

Chapter 2 Description of functions

1. ADMINISTRATION OF OPERATING CONTEXTS

The operating contexts are the software objects on which the software functions rely.

They contain static operating parameters, as well as a representation of the status at time t, which defines the actions possible on these contexts, enabling subsequent processing to be envisaged.

List of contexts:

- Contexts specific to the Software:
  - MCS instance,
  - MCS channel,
- Operating contexts handled by the Software and belonging to FIP DEVICE MANAGER:
  - FIP DEVICE MANAGER instance,
  - FIP DEVICE MANAGER channel,
- FIP DEVICE MANAGER message contexts:
  - the MCS instance enables the Software to be integrated into software tasks or processes,
  - the MCS channels provide the transport services offered by the SEGMENT_FDM software,
  - the FIP DEVICE MANAGER instance enables the FDM software to be integrated into software tasks or processes. An FDM instance is associated with a WorldFIP network,
  - the message contexts provide the data link transfer services offered by the FDM_R4 software.

Characteristics of FDM message contexts

- The SEND message contexts enable messages to be sent at DLL level from a local Source Address to a Destination Address.
- The RECEIVE message contexts enable messages to be received at DLL level from a Source Address at a local Destination Address.
- The FULLDUPLEX message contexts enable messages to be sent and received at DLL level.
- The FIP message channels enable a FIP message context to use a given part of the bandwidth for FIP network messages.

  These channels are periodic, when the bandwidth has an initial fixed size, or is aperiodic. They are needed for **SENDING** DLL messages.

Links between FDM and SEGMENT_FDM context:

- an instance of the Software is attached to one and only one instance of FDM, that is a unique WorldFIP node,
- a MCS channel is mapped on an FDM message context and this context is only handled by this MCS channel.

The various contexts are configured as follows

- It is up to the user to configure the WorldFIP channels at DLL level necessary for sending messages. (We talk of 'periodic' or 'aperiodic' WorldFIP channels), via the procedures offered by the FDM_R4 software.
- The Software offers the user a configuration interface for:
  - creating the general operating context: MCS instance,
  - configuring the MCS channels.

1.1 Creation of general operating context: MCS instance

Creation of the instance:

- leads to allocation and initialisation of all the necessary resources,
- identifies the user callback functions, which are to be called by the Software, to provide information about any faults encountered during operation,

  see Inform the user of operating faults
- identifies the user callback function which is to be called by the Software to inform the user that he must activate the MCS empty fifo task,

  see Sequence the actions to be performed
- identifies the user callback function which is to be called by the Software to inform the user of the result of the channel command,

  see Channel commands

1.2 Inform the user of operating faults

Checking user actions

All the user's actions are checked so that a situation is never reached in which the software's behaviour is non-deterministic.

These checks are usually run upstream, when the interface procedures are called by the user.

In any case, the user must be precisely informed of why the service he requests has been rejected: reports are provided immediately as well as later on.

When it can, the Software also gives information about problems encountered, using *Warnings* or *Fatal error*s or *Connection Failure errors* by calling user callback procedures which were supplied at creation of the instance.

Checking the configurations

When the configurations are created, the Software runs as many checks as possible in order to guarantee the correct working of the corresponding context, if creation is accepted.

The configuration parameters are carefully checked, so that only configurations which have a real meaning are accepted.

In addition, memory management checks are run, as well as checks on configuration objects on which the Software relies.

Warning of malfunctions on a channel

Any problem encountered by the Software is explicitly reported to the user.

By their very nature, A_DATA type channels must be capable of sending and receiving according to the capabilities initially assigned to them.

In particular, the acknowledgement, repetition/anti-duplication mechanisms must be able to absorb network transmission faults and carry out reliable transfers in reasonable time.

As soon as the Software detects a malfunction in a channel, it warns the user by calling a warning procedure giving the reference of the faulty channel.

The following types of malfunction are possible:

- message transfer failure on a transmission channel,
- detection of sequence break in the reception window, on a reception channel.

1.3 Sequencing the actions to be performed

The following principle is used:

- all the events, for which processing is not considered to be "short", are stored in interconnection queues.

These events are on hold until the Software takes over control, retrieves them one by one and initiates processing of them in turn.

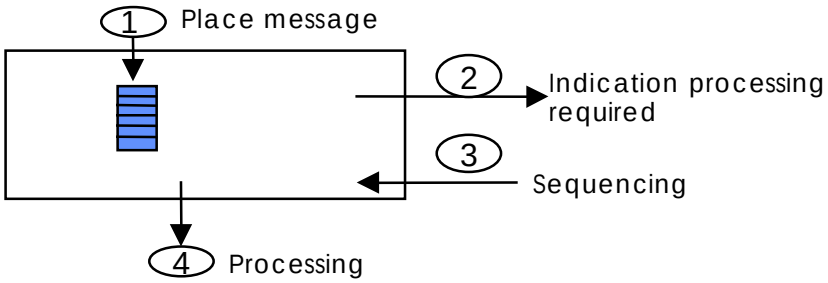
These events are as follows:

- channel command requests,
- message transfer requests,
- message transfer confirmations,
- reception of data link messages,
- expiry of timeouts.

The interconnection queues are as follows:

- channel command requests queue,
- message transfer request queue,
- "DLL frames" reception queue,
- link transfer confirmations queue ("DLL frame" sent acknowledgement queue),
- timeouts expiry queue. This queue contains the times for management of the "DLL frame" acknowledgement timeouts.

The operating model is the same as that used by FIP DEVICE MANAGER.



① A request is stored in the "interconnection queue".

② When an interconnection queue goes from empty to not-empty, the user is called by the ***Extern_Signal_MCS*** procedure. The user must write this procedure.

③ In this procedure, the user must activate a task enabling control to be given to the Software. When the software has control, via procedure **mcs_fifos_empty(),** it scans the interconnection queues. When one of them is other than empty, it extracts the message and triggers processing ④ by calling the appropriate procedure.

2. CHANNEL COMMANDS

The Software offers the user an interface for commanding a channel. There are 3 types of channel commands:

- creation request,
- stop request,
- destruction request.

These commands can only take place if the MCS instance initialisation phase has been carried out.

When the Software receives a channel command request, it

- checks the validity of the command (the sequencing on the channel commands is creation - stop – destruction).
- puts the request in a queue and when the software is activated it makes the following processing:
  - If the request is invalid, a report will be immediately returned.
  - If the request is a stop request, it stops the channel whose reference is given in the command is stopped (the message send/reception is stopped).
  - If the command is a destruction command, the channel whose reference is given in the command is destructed.

The user receives the confirmation of his request via the callback procedure given during the configuration of the MCS instance.

2.1 Channel configuration

The creation request contains the channel configuration parameters.

Each channel is configured for connection to:

- either a "MICROFIP" type station,
- or a "FULLFIP2" type station.

The message size that can be sent or received on these channels will be configured for each channel.

The remote station connected to the channel must have the same value.

All the MCS channels have the following properties:

- type A_DATA MCS channels: point to point, with PDU acknowledgement,
- anticipation = 1,
- number of repetitions = 1.

Each MCS channel has the following characteristics:

- the SDU MaxSDUSize that can be sent or received on a channel is configured. This size must have the same value on the remote station.

  Its maximum value = 0xFFFF, or 65535 bytes,

- the user gives the type of destination subscriber (MICROFIP or FULLFIP2): DLLType

  The size of the "DLL frame" differs depending whether one is connected to a MICROFIP type station (128 bytes) or a FULLFIP2 station (262 bytes).

  Internally, the Software thus configures:

  - DLL_Size=122: size of message data link layer in bytes, for MICROFIP subscriber,
  - DLL_Size=256: size of message data link layer in bytes, for FULLFIP2 subscriber,

- ConnexionTempoTicks is configured:
  - each message is segmented into x DLL frames. For each DLL frame sent there must be a corresponding DLL acknowledgement frame returned,
  - connexionTempoTicks is the value of the timeout which triggers repetition of a DLL frame when the channel has not received an acknowledgement for this frame.

The message contexts needed by FDM (LSAPs) are created by the Software during this configuration process,

- the type of message used msg_type is configured:

  Depending on the type used, the Software configures the WorldFIP channel associated with the message context:

  - 0: "aperiodic" channel, associated with the aperiodic message traffic,
  - 1: "periodic" channel, associated with a message identifier circulating periodically.

- Local_Address and Remote_Address are configured:
  - Local_Address: corresponds to source LSAP + segment number (source address + segment),
  - Remote_Address: corresponds to destination LSAP + segment number (destination address + segment).

Note

One and only one address is used per channel.

If the remote station is a MICROFIP station, you must configure Remote_Address = 00 xy zz

(xy = physical address of the MICROFIP station on the network
zz = segment number).

- Configure *User_Msg_Rec_Proc()*:
  pointer to the callback function which is called by the Software to send the user the message reception procedure message,

- Configure *User_Msg_Ack():*
  pointer to the callback function which is called by the Software to send the user the message sent acknowledgement.

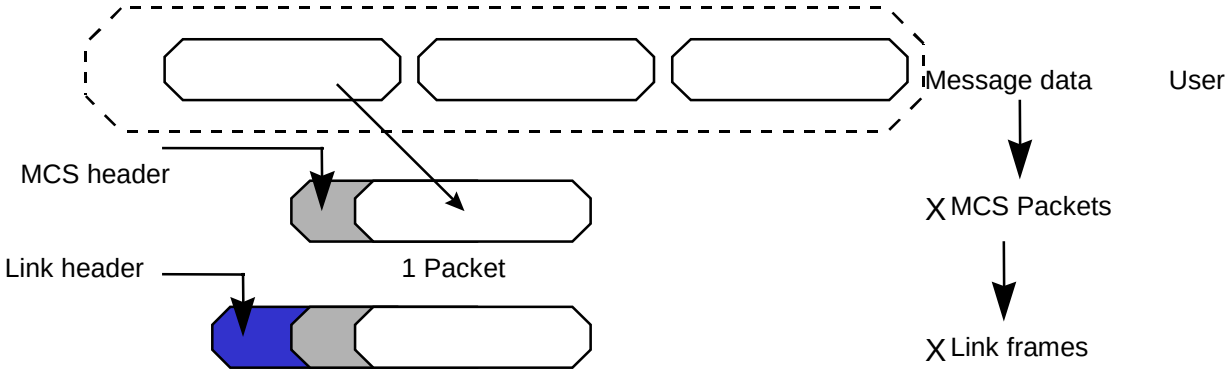
3. TRANSFERS

Data transfers take place via A_DATA type channels in both SEND and RECEIVE directions.

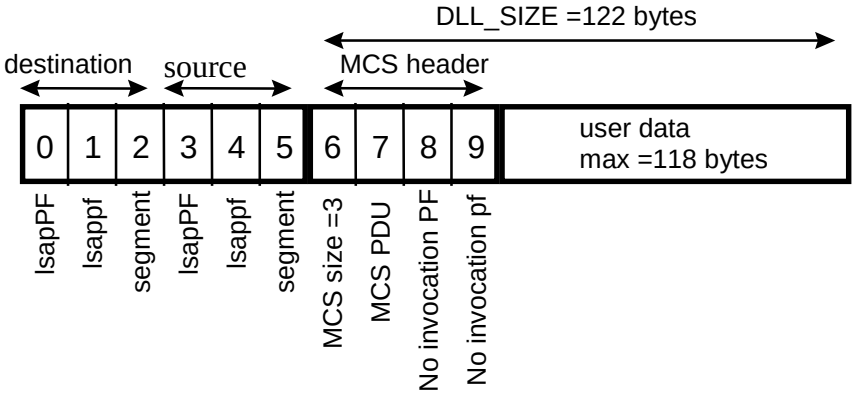
The events which activate the transfers are transmission requests from the user and reception of data units from the network.

Transmission and reception of messages using the A_DATA protocol takes a certain amount of time, as it implements mechanisms such as segmentation and reassembly, with acknowledgement of each data unit:

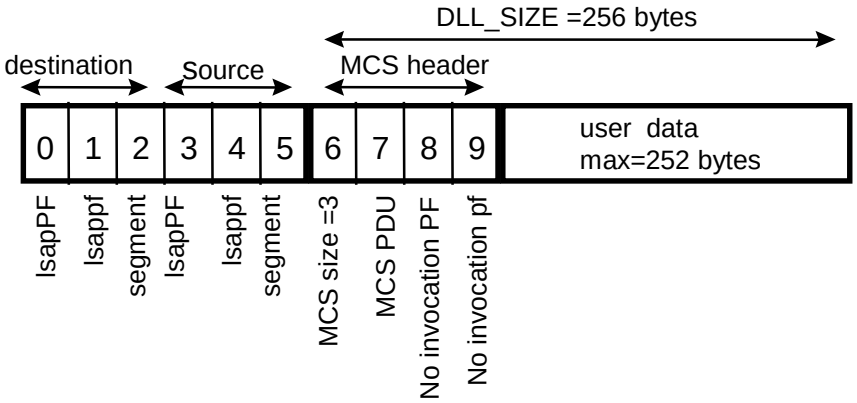
- when sending a user message, segmentation consists in chopping the message into packets,
- when receiving "DLL frames", reassembly consists in combining them to create a message,
- **Packet**: user data entity with associated MCS header, transmitted to the data link layer,
- **Frame**: data unit transmitted on the physical layer.



For exchange on a channel connected to a MICROFIP type station:



For exchange on a channel connected to a FULLFIP2 type station



The MCS header comprises a PDU giving the type of frame:

- start PDU (0x24),
- middle PDU (0x25),
- end PDU (0x26),
- complete PDU (0x27) when the SDU can fit into a single frame.

Each PDU sent must be acknowledged by an Ack. PDU (0x60), frame without user data.

When the channel does not exist, the transfer request is refused and an immediate negative report is produced and the Software notifies it using the Warning procedure.

3.1 Sending a message on a channel

When the Software receives a send message request from the user:

- it checks that the channel on which the user asks to send the message has indeed been configured and if the channel is in the running state (not stopped or in destruction state).

If not, the request is qualified as negative, with the appropriate report;

- it stores the message in the queue and when the Software has control, it carries out the following:
  - segments the message to be sent into x "DLL frames",
  - sends the "DLL frames", managing acknowledgement and repetition,
  - gives the user confirmation of transfer,
  - warns the user of any malfunction.

In the configuration of a channel, the number of requests in parallel was set at = 1, so the user must receive confirmation of message transfer before being able to send a new message.

The user receives confirmation of transfer via the callback procedure which he provided at configuration of each channel.

This confirmation is valid if each "DLL frame" making up the message has been sent and received an acknowledgement "DLL frame".

If problems occur, the user is informed by connection fault Warning or Signal procedures.

3.2 Receiving a message on a channel

FIP DEVICE MANAGER differentiates between the network frames according to the destination addresses.

The "DLL frames" are sorted by the software and if they make up a message:

- with start, middle or end MCS PDU or,
- with start-middle_end PDU MCS,

then the Software sends a "DLL frame" with acknowledgement PDU and reassembles the "DLL frames".

When the message is complete, the user is notified, by the callback procedure which he provided at channel configuration, that a message has been received on the channel concerned.

The data making up the message are supplied in x blocks.

The user can call a primitive supplied by the Software, in order to reconcatenate the message received into a single block.

The user is notified of a message number and the total size of the message data.

It is up to the user to release the SDU received, as the Software has to manage recycling of memory resources.

If problems occur, the user is informed by connection fault Warning or Signal procedures.

Chapter 3 Primitives

1. DESCRIPTION OF USER PRIMITIVES

1.1 Procedure mcs_get_software_version

Prototype

mcs_get_software_version(MCS_VERSION * Version)

Description

This procedure is used to identify the MCS software version used.

Input parameters

Version: pointer to the storage zone for the MCS_VERSION type version read.

Definition of type: MCS_VERSION

```
typedef struct {
    char  Version;
    char  Revision;
    char  Proto;
    char  ProtoNr;
    char  Date[3];

} MCS_VERSION;
```

Version: version no.

Revision: revision no.

Proto; = 1 => if prototype version = 0 => official version

ProtoNr: prototype number, only significant if Proto =1

Date[0]: Year

Date[1]: Month

Date[2]: Day

Output parameters

None

1.2 Procedure mcs_initialize

Prototype

MCS_REF *mcs_initialize (MCS_CONFIGURATION *param

_MCS_ERROR_CODE_LIST *report)

Description

This USER side interface is used to create an instance of the Software, in other words, to initialise the operating context of this instance.

Creating an instance leads to allocation and initialisation of all the resources required for the working of this instance.

Input parameters

param: pointer to a table containing the configuration to be initialised by the MCS_CONFIGURATION type user table.

report: of type enum _MCS_ERROR_CODE_LIST contains the error code for the reason for the failure to create the MCS instance (see chapter 5).

Definition of type: MCS_CONFIGURATION

```
typedef struct {
    char        *Name;
    FDM_REF     *fdm_ref;
        void    (*Extern_Signal_MCS) ();
        void    (*Extern_Signal_Error) (MCS_TYPE_ERROR_WARNING,
                void * ,_MCS_ERROR_CODE_LIST);
        void    (*Extern_Signal_Warning ) (MCS_TYPE_ERROR_WARNING,
                void * ,_MCS_ERROR_CODE_LIST);
        void    (*Extern_Signal_Command_Cnf) (struct _MCS_REF *,
                struct _MCS_CHANNEL_COMMAND *);
        void    (*Extern_Signal_ConnexionFailure) ( USER_MCSCHANNEL_REF ,
                  _CONNEXION_FAILURE_LIST)
                MEMORY_RN   *Memory_Management_Ref;

}MCS_CONFIGURATION;
```

In italics:	procedures to be supplied by the user: see chapter 3 Subsection 1.1.9.
Name:	name given to the instance: maximum 8 characters
fdm_ref:	reference of the FDM instance to which the SEGMENT_FDM instance is attached.
Extern_Signal_MCS:	pointer to a user function to be called by SEGMENT_FDM to tell it to activate the void task.
Extern_Signal_Error:	pointer to a user function to be called by SEGMENT_FDM to indicate a serious software error.
Extern_Signal_Warning:	pointer to a user function to be called by SEGMENT_FDM to indicate a minor software error.
Extern_Signal_ConnexionFailure:	pointer to a user function to be called by SEGMENT_FDM to indicate a connection failure.
Extern_Signal_Command_Cnf:	pointer to a user function to be called by SEGMENT_FDM to inform the user about the command channel result.
Memory_Management_Ref:	Pointer to a structure which is necessary, when the partitioned memory manager is used. Must be initialised at NULL_PTR, of not partitioned memory manager is used. MEMORY_RN is a structure defined in the file gdm.os.h of the software FDM_R4. This file is to complete depending on user OS.

Output parameters

MCS_REF: pointer to MCS_REF type instance created

Pointer NULL if creation was impossible: the error code is then set in the report parameter.

Definition of type: MCS_REF

Software internal: not handled by the user

1.3 Procedure mcs_fifo_empty()

Overview

Activation of the void MCS fifos task **after general initialisation** (mcs_initialize()).

This activation should be triggered when the user has received the signal via the Extern_signal_MCS() procedure.

Prototype

void mcs_fifos_empty(MCS_REF * Mcs);

Description

This function enables the user:

- to activate a command channel, for which the request was transmitted by the primitive mcs_channel_command(),
- to activate sending of a message on a channel, for which the request was transmitted by the primitive mcs_send_message(),
- to identify the message sent acknowledgement, with the Software activating user procedure *User_Msg_Ack()*,
- to identify the messages received, with the Software activating user procedure *User_Msg_Rec_Proc()*.

Note

User_Msg_Ack():	user procedure to acknowledge transmission on a channel,
User_Msg_Rec_Proc:	user procedure for reception of message on a channel,
User_Msg_Ack() *User_Msg_Rec_Proc:*	two procedures given by the user in creation command channel.

Input parameters

Mcs: pointer to the reference of type MCS_REF

This is the pointer which was supplied during the initialisation procedure mcs_initialize

Output parameters

None

1.4 Procedure mcs_get_errorandwarningstring

Prototype

void msc_get_errorandwarningstring (char * String,_MCS_ERROR_CODE_LIST Code)

Description

This user help procedure gives the user a clear report associated with type MCS_ERROR_CODE_LIST.

It can be used when the user wishes to print the code plainly (rather than just a number).

Input parameters

String: pointer to code storage zone. This zone supplied is assumed to have at least 40 characters.

Code: error code to be translated

1.5 Procedure mcs_channel_command

Prototype

_MCS_ERROR_CODE_LIST mcs_channel_command (MCS_REF *Mcs,
MCS_CHANNEL_COMMAND *mcschannelcmd);

Description

This procedure is used to create, stop and delete a channel.

There is a sequence for a channel:

- You create a channel.
- You can stop only a created channel.
- You can delete only a stopped channel.

You must give the input parameters until the user callback procedure *Extern_Signal_Command_Cnf*() is called. This callback provides the request result.

Input parameters

mcs_ref: pointer to the MCS_REF instance

mcschannelcmd: command request, of type MCS_CHANNEL_COMMAND

Definition of type: MCS_CHANNEL_COMMAND:

```
typedef struct _MCS_CHANNEL_COMMAND{
    struct _MCS_CHANNEL_COMMAND          *Next;
    struct _MCS_CHANNEL_COMMAND          *Previous;
    struct _MCS_REF                      *mcs_ref;
    _MCS_CHANNEL_COMMAND_LIST            Command;
    MCS_USER_CHANNEL_CONFIGURATION       *paramIN_for_create;
    struct _MCSCHANNEL_REF               *channelmcs;
    _MCS_ERROR_CODE_LIST                 Report;
    }MCS_CHANNEL_COMMAND;
```

Note

This type (and certain types on which it depends) contain private fields, the use and content of which do not concern the user. *These fields are in italics*.

Command: command type: create, stop, delete (see _MCS_CHANNEL_COMMAND _LIST)

paramIN_for_create: pointer to the configuration channel to create (see MCS_USER_CHANNEL_CONFIGURATION)

channelmcs: see MCSCHANNEL_REF

Report: field used when confirming the request (see *Extern_Signal_Command_Cnf*())

Definition of type: enum_MCS_CHANNEL_COMMAND_LIST:

```
typedef enum{
_MCS_CHANNEL_CREATE,
_MCS_CHANNEL_STOP,
_MCS_CHANNEL_DELETE
};
```

Description of type MCSCHANNEL_REF

Software internal: not handled by the user:

Reference of an MCS channel created.

The message is sent/received on a channel reference.

Definition of type: MCS_USER_CHANNEL_CONFIGURATION:

```
typedef struct _MCS_USER_CHANNEL_CONFIGURATION{
    USER_MCSCHANNEL_REF   UserChannelRef;
    unsigned long         Local_Address;
    unsigned long         Remote_Address;
    unsigned short        ConnexionTempoTicks;
    unsigned short        MaxSDUSize;
    _MCS_TYPE_DLL         DLLType;
    void                  (*User_Msg_Rec_Proc) (USER_MCSCHANNEL_REF,
                          struct _MCS_SDU_RECEIVED *);
    void                  (*User_Msg_Ack) (USER_MCSCHANNEL_REF,
                          struct _MCS_SDU_TO_SEND *);
    _MCS_MSG_TYPE         msg_type;
    _MCS_MSG_MODE_ACK     msg_mode_ack;
};
```

UserChannelRef: user reference: one per MCS channel

Local_Address: address Local_DLL_Address (message transmission source Lsap + segment number)

Remote_Address: address Remote_DLL_Address (message transmission destination Lsap + segment number)

ConnexionTempoTicks: number of "ticks" needed to trigger a time_out:

Timeout value triggering repetition of a data PDU sent following a user's send message request, when the channel has not received an acknowledgement for this PDU. (one tick = an fdm_ticks_counter() call, see manual ALS 50278)

MaxSDUSize: maximum size of message being sent or received on the channel

DLLType: type of exchange at DLL level

- connection channel to a FULLFIP2 type subscriber,
- connection channel to a MICROFIP type subscriber.

User_Msg_Rec_Proc: Pointer to a user function to be called by the Software after a message is received

User_Msg_Ack : Pointer to a user function to be called by the Software after a message acknowledgement is received

Note

The callback procedures can be the same for all the channels.

msg_type: Type of message: periodic (on channel 1) or aperiodic

msg_mode_ack: acknowledgement mode for sending on the channel:

- sending in normalised mode (address acknowledged or not acknowledged)
- sending in non-acknowledged mode

Note

The non-acknowledged mode is preferable because the MCS acknowledgement frames are used.

Description of type: _MCS_TYPE_DLL

```
typedef enum    { _MCS_TYPE_DLL_FULLFIP,
                  _MCS_TYPE_DLL_MICROFIP
                        }
```

Description of type: _MCS_MSG_TYPE DLL

```
typedef enum    { _MCS_MSG_TYPE_PERIOD    periodic type messages
                  _MCS_MSG_TYPE_APERIOD   aperiodic type messages
}
```

Description of type: _MCS_MSG_MODE_ACK

typedef enum {

_MCS_MSG_MODE_NORMALISE,

_MCS_MSG_MODE_NOACK

}

Command	**mcs_channel_command**	***Extern_Signal_Command_Cnf*()**
_MCS_CHANNEL_CREATE	ParamIN_for_create	Channelmcs: channel reference of created channel NULL if not created
_MCS_CHANNEL_STOP	ParamIN_for_create: not use Channelmcs: channel reference of channel to stop	
_MCS_CHANNEL_DELETE	ParamIN_for_create: not use Channelmcs: channel reference of channel to delete	

Output parameters

_MCS_ERROR_CODE_LIST

_MCS_OK: if the request is accepted, the user will have the report of his command in the callback procedure *Extern_Signal_Command_Cnf()* or following error code:

- _MCS_CHANNEL_COMMAND_ON_ILLEGAL_INSTANCE
- _MCS_CHANNEL_COMMAND_ILLEGAL_PARAMETERS
- _MCS_CHANNEL_COMMAND_ILLEGAL_SERVICE
- _MCS_CHANNEL_COMMAND_ILLEGAL_CHANNEL_STATE

Note

A channel is in "running state" when he is created and if there is no stop or delete command on his reference.

You must give a stop command before a delete command

WARNING

Be careful, when you stop a channel: No receive or send message must be running.

1.6 Procedure mcs_send_message ()

Prototype

_MCS_ERROR_CODE_LIST mcs_send_message (MCS_CHANNEL_REF *channelmcs,
MCS_SDU_TO_SEND *msg)

Description

This procedure is run by the user when he wishes to send a message on a running channel.

The requested channel must exist and have been configured.

Input parameters

channelmcs: MCS channel reference. It was given at channel configuration.

msg: message to be sent, of type MCS_SDU_TO_SEND

Definition of type: MCS_SDU_TO_SEND

```
typedef struct _MCS_SDU_TO_SEND {
struct _MCS_SDU_TO_SEND                 *Next;
        struct _MCS_SDU_TO_SEND         *Previous;
        ALL_MCSCHANNEL_REFS             Channel;
        unsigned long                   Destination;
        unsigned int                    Nr_Of_Blocks;
        DESC_BLOCK_DATA_T               *Ptr_Block;
        MCS_SDU_TO_SEND_PRIVATE_AREA    Private_MCS;
        _MCS_ERROR_CODE_LIST            Service_Report;
        }
```

Note

This type (and certain types on which it depends) contains private fields, the use and content of which do not concern the user. *These fields are in italics*.

Nr_Of_Blocks: number of DESC_BLOCK_DATA_T type blocks in the message

Ptr_Block: pointer to the first message description block

Report: field used when confirming transmission acknowledgement

Definition of type: DESC_BLOCK_DATA_T:

Note

This type is the same as that defined by FDM_MSG_T_DESC in FDM_R4.

```
typedef struct _DESC_BLOCK_DATA_T {
        struct _DATA_BLOCK_TO_TRANSFER {
                unsigned short                  Nr_Of_Bytes;
                unsigned char                   *Ptr_Data;
                struct _DESC_BLOCK_DATA_T *Next_Block;
                } User_Block;
        struct _DATA_BLOCK_TO_TRANSFER Private;
} DESC_BLOCK_DATA_T;
```

Note

This type contains a private field, the use and content of which do not concern the user. *This field is in italics.*

Nr_Of_Bytes: Number of bytes contained in the block pointed by Ptr_Data

Ptr_Data: Pointer to the zone containing the data

Next_Block: Pointer to the next block of the same type

Output parameters

code of type _MCS_ERROR_CODE_LIST: See ERROR MESSAGES APPENDIX

_MCS_OK if the request is accepted

or following error code

_MCS_SEND_DATA_ON_INEXISTING_CHANNEL

_MCS_SEND_DATA_ON_IDLE_CHANNEL

_MCS_INTERNAL_ERROR

If the request is not accepted, then a *Extern_Signal_Error*() or *Extern_Signal_Warning*() procedure is triggered with the same error code.

Note

When the request is accepted, the following calls are possible:

either *Extern_Signal_Error*() which is triggered in the event of a major problem,

or *Extern_Signal_Warning* () which indicates a fault of type _MCS_ SEND_DLL_xxx,

on the reference of the channel concerned.

This warning is associated with the Service_Report field in MCS_SDU_TO_SEND which will be set to _MCS_SEND_DATA_DLL_FAIL when calling the transmission acknowledgement primitive.

1.7 Procedure mcs_free_received_message()

Prototype

void mcs_free_received_message (MCS_SDU_RECEIVED *msg);

Description

This procedure is to be run by the user, when he has received and analysed a message. It enables the Software internal resources created for this message to be released.

Following this procedure, all the data concerning the message is deleted.

Input parameters

msg: pointer to message to be released, of type MCS_SDU_RECEIVED

Definition of type: MCS_SDU_RECEIVED

```
typedef struct _MCS_SDU_RECEIVED {

struct _MCS_SDU_RECEIVED        *Next;

        struct _MCS_SDU_RECEIVED    *Previous;

        ALL_MCSCHANNEL_REFS         Channel;

        unsigned long               Source;

        unsigned short              SDU_Nr;

        unsigned short              SDU_Size;

        unsigned int                Nr_Of_Blocks;

        DESC_BLOCK_DATA_R           *Ptr_Block;

 } MCS_SDU_RECEIVED;
```

Note

This type contains private fields, the use and content of which do not concern the user. *These fields are in italics.*

SDU_Nr: number of the SDU

SDU_Size: total size of message received

Nr_Of_Blocks: number of blocks of type DESC_BLOCK_DATA_R

Ptr_Block: pointer to the first message description block

Definition of type: DESC_BLOCK_DATA_R

```
typedef struct _DESC_BLOCK_DATA_R {
unsigned short                  Nr_Of_Bytes;
unsigned char                   *Ptr_Data;
struct _DESC_BLOCK_DATA_R       *Next_Block;
}
```

Nr_Of_Bytes: number of bytes contained in the block pointed by Ptr_Data

Ptr_Data: pointer to the zone containing the data received

Next_Block: Pointer to the next block of the same type

Output parameters

None

1.8 Procedure mcs_cat_msg_received()

Prototype

void mcs_cat_msg_received (char *ptr_data_user,MCS_SDU_RECEIVED *msg);

Description

This procedure is a user help procedure.

When the user receives a message, it is of the type MCS_SDU_RECEIVED: this type consists of a set of x description buffers. This procedure enables the user to reconcatenate all the data into a single block on his own system.

Input parameters

ptr_data_user: pointer to a buffer that the user must allocate:

The user knows the size to be allocated for this buffer: that given in field SDU_Size of MCS_SDU_RECEIVED

msg: pointer to received message to be concatenated, of type MCS_SDU_RECEIVED

Output parameters

None

1.9 Primitives called, to be supplied by the user

Summary of procedures to be written by the user and called by the Software to inform the user of various events.

The user can change the following procedure names:

PROCEDURE	Reference sent to the Software by:	Called by the Software to report
Extern_Signal_Error()	mcs_initialize() in parameter of type MCS_CONFIGURATION	detection of serious errors
Extern_Signal_Warning()	mcs_initialize() in parameter of type MCS_CONFIGURATION	minor errors
Extern_Signal_ConnexionFailure	mcs_initialize() in parameter of type MCS_CONFIGURATION	Detection of connection failure
Extern_Signal_MCS()	mcs_initialize() in parameter of type MCS_CONFIGURATION	that the user must activate task mcs_fifos_empty()
Extern_Signal_Command_Cnf	mcs_initialize() in parameter of type MCS_CONFIGURATION	acknowledgement of channel command
User_Msg_Rec_Proc() (one per configured channel)	mcs_createandstart_channels() in parameter of type MCS_USER_CHANNEL_CONFIGURATION	reception of a message on a channel
User_Msg_Ack (one per configured channel)	mcs_createandstart_channels() in parameter of type MCS_USER_CHANNEL_CONFIGURATION	acknowledgement of message sent on a channel

Prototype of procedures:

void (**Extern_Signal_MCS*) (void);

void (**Extern_Signal_Error*) (MCS_TYPE_ERROR_WARNING p1,

void * p2 ,_MCS_ERROR_CODE_LIST p3);

void (**Extern_Signal_Warning*) (MCS_TYPE_ERROR_WARNING p1,

void *p2 ,_MCS_ERROR_CODE_LIST p3);

void (**Extern_Signal_ConnexionFailure*) (USER_MCSCHANNEL_REF,
_CONNEXION_FAILURE_LIST);

void (**Extern_Signal_Command_Cnf*) (struct _MCS_REF *,_MCS_CHANNEL_COMMAND *);

void *User_Msg_Rec_Proc* (USER_MCSCHANNEL_REF, struct _MCS_SDU_RECEIVED)

void *User_Msg_Ack* (USER_MCSCHANNEL_REF, struct _MCS_SDU_TO_SEND)

Description of types:

_MCS_ERROR_CODE_LIST: gives the error code list (see chapter 5)

Description of type: _MCS_TYPE_ERROR_WARNING

This type is used to identify the error or warning source, in order to find out the type of parameter P2.

typedef enum {

_MCS_TYPE_MCS_CHANNEL_CONFIGURATION,

_MCS_TYPE_MCS_SDU_TO_SEND,

_MCS_TYPE_FDM_MSG_RECEIVED,

_MCS_TYPE_FDM_MSG_TO_SEND,

_MCS_TYPE_MCS_FDM_MSG_TO_SEND,

_MCS_TYPE_MCS_CHANNEL_REF

}

As regards the user, it is mainly field p3 that has to be analysed.

For certain warnings or errors, it is however of interest to find out the reference of the faulty channel. This reference can be read in parameter P3 when parameter type P1= _MCS_TYPE_MCS_CHANNEL_REF
(see Error messages chapter: code _MCS_ERROR_CODE_LIST)

The other cases are used to identify the Software level at which the error occurred.

Description of type: _CONNEXION_FAILURE_LIST

(see chapter “Error messages”: code _CONNEXION_FAILURE_LIST)

2. DESCRIPTION OF OS PRIMITIVES

2.1 OS primitives of FDM_R4

The user must fill out the OS primitives as defined in manual ALS 50278.

The primitives concerning some OSs are already filled out.

If the user uses an OS other than that defined in the FDM_R4 compilation options (see file User_opt.h), he must then write the procedures concerning his OS in files **fdm_os.h** and **fdm_os.c.**

2.2 OS primitives for SEGMENT_FDM

The Software uses the same FDM_R4 files User_opt.h, fdm_os.h and fdm_os.c.

If the user uses an OS other than that defined in the FDM_R4 compilation options (see file User_opt.h), he must then write the procedures concerning his OS in file **mcs_os.c**

This concerns the following procedures:

BinarySemaphoreCreate(): creation of a binary semaphore

SemaphoreDelete() : deletion of the binary semaphore

Chapter 4 Software initialisation

1. HOW TO RUN THE SOFTWARE

The actions involved must be run in a certain order, as described below.

1. Initialise the FDM_R4 software:
   - configure the DLL (transfer rate, physical address, segment number etc.),
   - initialise the FDM_R4 message part,
   - configure the WorldFIP channels for transmission (locally create the WorldFIP identifiers, providing the FIP node with the message bandwidth).
2. Run an appropriate BUS ARBITER program.
3. Following this initialisation, you must start the Software initialisation.

2. INITIALISE FDM_R4

The Software uses the message services of FIP DEVICE MANAGER.

Therefore, before using the Software, you must have made all the calls needed for operation of the FDM_R4 (see manual ALS 50278, chapter 1 Section 4: **"**Overview of interface with user application"):

- creation of the FDM device,
- creation and start-up of the network (by correctly configuring all the message related information),
- management of internal timeouts,
- activation of message transmission function,
- activation of message reception function.

2.1 Use of the Periodic Message system

For FDM_R4, we use periodic channel No. 1.

So that the periodic message system works, you must declare the WorldFIP identifier (one identifier per station) used for this channel.

The declaration procedure is an FDM_R4 user function:

fdm_channel_create ()

Note

See condition on the BA to ensure that the periodic message system works.

2.2 Use of the Aperiodic Message system

The channel used for aperiodic messages is channel 0.

So that the aperiodic message system works, the station must produce at least one MPS variable with message request authorisation (see FDM R4).

Note

See condition on the BA to ensure that the aperiodic message system works.

3. RUN AN APPROPRIATE BUS ARBITER PROGRAM

BA for the periodic message system:

For the periodic message system to work, you must periodically circulate the identifier of the periodic channel of each station (ID_MSG, identifier No.).

For the aperiodic message system:

For the aperiodic message system to work, you must periodically circulate the identifier associated with production of the MPS variable with authorisation, and an aperiodic message window of correct size is required.

(ID_DAT, identifier_no)

BA part for use with a MICROFIP HANDLER Station

So that the messages are accepted by a MICROFIP HANDLER station, the ID_MSG 06xy must be circulated, where xy = station physical address, with each station having a different physical address.

4. INITIALISE THE SOFTWARE AND CONFIGURE THE "MCS CHANNELS"

The Software is initialised when the operator calls the function:

mcs_initialize()

Once the action described above has been carried out, the services are available, in other words, the **USER** can:

- ask to command channels,
- when a channel is running (after acknowledgement of create channel command):
  - ask to send a message on a running channel,
  - receive messages and message sent confirmations on a running channel,
- when the user wants to stop and delete a channel he must check that: no send or receive message is running.

Sequence to be followed by the user when receiving a message

1. The user is warned of message reception when the call-back procedure *User_Msg_Rec_Proc()* associated with the context is called.
2. The user then analyses the message received, the message being in the form of n blocks. He can concatenate the whole message on his own system:
   To do this, he must allocate a contiguous zone on his system, the size of which is given in the received message structure. He then calls the primitive *mcs_cat_msg_received()*.
3. When the user has received the message, he must call the primitive *mcs_free_received_message()*, in order to release the resources used for message reception: error messages.

Chapter 5 Error messages

1. CODE _MCS_ERROR_CODE_LIST;

This code is a list.

It is used:

- either in the immediate procedure report → reported by (**CR**) in the following table,
- or in the warning or error procedures:
  - Extern_Signal_Error; → reported by **E** in the following table,
  - Extern_Signal_Warning → reported by **W** in the following table.

Value	Mnemonic	Context	E/W CR	
0	_MCS_NOK	Indicates requested operation successful		
1	_MCS_NOK,	Indicates error in requested operation		
2	_MCS_INTERNAL_ERROR	Internal Software error: SERIOUS	E	
	_MCS_CREATE_xxx faults	Error detected during procedure mcs_initialize()		
0x10	_MCS_CREATE_ILLEGAL_PARAMETER	Illegal input parameter	CR	
0x11	_MCS_CREATE_ALLOCATE_MEMORY_FAULT,	Memory allocation problem	CR	
0x12	_MCS_CREATE_CREATE_SEMAPHORE_FAULT,	Semaphore creation problem	CR	
	MCS CHANNEL_CREATE_xxx faults	Error detected during procedure mcs_createandstart_channels()		
0x13	_MCS_CHANNEL_CREATE_ON_ILLEGAL_INSTANCE	Instance not valid, must give that supplied by mcs_initialize()	CR	
0x14	_MCS_CHANNEL_CREATE_ILLEGAL_USERCHANNEL_REF	A user reference in table MCS_USER_CHANNEL_CONFIGURATION is not valid: It must be unique to each MCS channel	CR	
0x15	_MCS_CHANNEL_CREATE_TOO_MUCH_CHANNELS	Number of channels >256	CR	
0x16	_MCS_CHANNEL_CREATE_ALLOCATE_MEMORY_FAULT	Memory allocation problem	CR	
0x1B	_MCS_CHANNEL_CREATE_ILLEGAL_CALL_BACK_ FUNCTION	A call_back User_msg_ack or User_msg_rec procedure is null	CR	
0x1C	_MCS_CHANNEL_CREATE_ILLEGAL_LOCAL_ADDRESS	Error on a local LSAP	CR	
0x1D	_MCS_CHANNEL_CREATE_ILLEGAL_REMOTE_ADDRESS	Error on a remote LSAP	CR	
0x1E	_MCS_CHANNEL_CREATE_ILLEGAL_CHANNEL_NR	Error on Channel_NR: channel should be <9	CR	

Value	Mnemonic	Context	E/W CR	
0x1F	_MCS_CHANNEL_CREATE_ILLEGAL_SDU_SIZE	SDU size >0xFFFF	CR	
0x22	_MCS_CHANNEL_CREATE_ILLEGAL_REFUSED_DLL_CTXT _CREATION	Creation of message context refused by FDM_R4	CR	
	MCS SEND_DATA _xxx faults	Errors detected at message transmission *mcs_send_message*()		
0x100	_MCS_SEND_DATA_ON_INEXISTING_CHANNEL	Reference of channel on which message sent does not exist	E CR	Note 1
0x104	_MCS_SEND_DATA_INVALID_LENGTH	Message size not valid: Size = 0 or size > Max_SDU_size configured for the channel	W CR	Note 1
0x106	_MCS_SEND_DATA_DLL_FAIL	Cause: Error during transfer to DLL layer in Report field: see callback procedure User_msg_ack()		
	MCS RECEIVE _xxx faults	Errors detected at message reception by DLL layer		
0x200	_MCS_RECEIVE_DLL_ON_INEXISTING_CHANNEL	This error is indicated in Extern_Signal_error. Problem with MCS channel reference	E	Note 2
0x201	_MCS_RECEIVE_DLL_ON_IDLE_CHANNEL	This error is indicated in Extern_Signal_warning. Problem with MCS channel reference: the reference exists, but the MCS channel is not in a normal state	W	Note 2
0x203	_MCS_RECEIVE_DLL_INVALID_PDU	This error is indicated in Extern_Signal_warning. Reporting an error in PDU MCS	W	Note 2
0x204	_MCS_RECEIVE_DLL_BROKEN_MESSAGE	This error is indicated in Extern_Signal_error. Reporting incorrect message received from DLL	E	Note 2
	MCS CHANNEL_CONNECTED_xxx faults	Errors		
0x400	_MCS_CHANNEL_CONNECTED_LACK_OF_MEMORY_FOR_ USER_IND	Problem with allocating memory resources at message reception by DLL layer	W	Note 5
	MCS SEND_DLL_xxx faults	Errors detected at frame transmission (part of SDU) on the DLL layer. These warnings are associated with _MCS_SEND_DATA_DLL_FAIL		
0x500	_MCS_SEND_DLL_ON_INEXISTING_CHANNEL	the FIP message channel does not exist (we are sending on a periodic channel which was not created by fdm_create_canal())	W	Note 5
0x501	_MCS_SEND_DLL_ON_IMAGE2,	The message system is not configured on IMAGE2 and we are asking to send on this image	W	Note 5
0x502	_MCS_SEND_DLL_INTERNAL_ERROR,	Internal error detected in the frame sent on DLL layer (data size =0 ……)	W	Note 5
0x503	_MCS_SEND_DLL_UNKNOWN_ERROR,	Unknown error detected in frame sent on DLL layer	W	Note 5

Reminder:

Procedures *Extern_Signal_Error* and *Extern_Signal_Warning* include parameters P1 and P2 (see chapter dealing with Primitives to be supplied by the user: procedure prototypes).

The error codes above are associated with parameters P1/P2 defined in the following notes:

Note	**Parameter P1: List** **_MCS_TYPE_ERROR_WARNING**	**Parameter P2** **Pointer to**
1	_MCS_TYPE_MCS_SDU_TO_SEND	MCS_SDU_TO_SEND
2	_MCS_TYPE_FDM_MSG_RECEIVED	FDM_MSG_RECEIVED
3	_MCS_TYPE_FDM_MSG_TO_SEND	FDM_MSG_TO_SEND
4	_MCS_TYPE_MCS_FDM_MSG_TO_SEND	MCS_FDM_MSG_TO_SEND
5	_MCS_TYPE_MCS_CHANNEL_REF	MCS_CHANNEL_REF

Note 1: _MCS_TYPE_MCS_SDU_TO_SEND

This type is used in *Extern_Signal_Error* or *Extern_Signal_Warning* following calling of procedure mcs_send_message() as well as in its immediate report.
It indicates an error on input parameter MCS_CHANNEL_REF * for this procedure.

Note 2: _MCS_TYPE_FDM_MSG_RECEIVED

This type is used in *Extern_Signal_Error* or *Extern_Signal_Warning* following processing of an indication from FDM_R4.
It indicates an error in input parameter MCS_CHANNEL_REF * for this procedure.

Note 3: _MCS_TYPE_FDM_MSG_TO_SEND

This type is used in *Extern_Signal_Error* following calling of procedure mcs_send_message() and means that the FDM message context associated with the MCS channel no longer exists, which is impossible.

Note 4: _MCS_TYPE_MCS_FDM_MSG_TO_SEND

This type is used in *Extern_Signal_Error* following an acknowledgement frame transmission error.

Note 5: _MCS_TYPE_MCS_CHANNEL_REF

This type is used to give the references of the MCS channel in *Extern_Signal_Error* or *Extern_Signal_Warning.*

2. CONNEXION_FAILURE

This code is a list. It is used in the procedure *Extern_Signal_ConnexionFailure*:

void (**Extern_Signal_ConnexionFailure*) (USER_MCSCHANNEL_REF,
_CONNEXION_FAILURE_LIST);

Value	Mnemonic	Context
0	_CONNEXION_FAILURE_In_ProtocoleExchange	Exchange error: the received PDU MCS number is not the expected one.
1	_CONNEXION_FAILURE_BrokenSequence	Sequence number error when a MCS frame was received
2	_CONNEXION_FAILURE_SDUTransferError	Error when a message is transferred Not all the acknowledgement PDUs on all the sent frames are available (In this case the acknowledgement of the sent message is negative in *User_Msg_Ack*)
3	_CONNEXION_FAILURE_PDUTransferError	After the transfer of a PDU no PDU acknowledgement has been received.

Chapter 6 Installation procedures

1. PRESENTATION OF DELIVERY

SEGMENT_FDM source files:

mcs_user.c	mcs.h
mcsconfi.c	mcs_os.h
mcs_os.c	mcsprive.h
mcstools.c	mcspriv2.h
mcsheart.c	mcs_opt.h

Note

The user must obtain the FDM_R4 source files.

2. IMPLEMENTATION

For this part, you should refer to Manual ALS 50278 and:

- write the content of the user procedures likely to be used,
- fill out the OS interface macros:
  - file fdm_os.c (FDM_R4),
  - file mcs_os.c (SEGMENT_FDM),
- customise file user_opt.h (FDM_R4) according to the needs of the application concerned,
- compile files XXX.c with the appropriate compiler, using the relevant compilation options,
- build library FIPMAN.lib (this name simply an example), using the appropriate library manager,
- build library SEGMENT_FDM.lib (this name simply an example), using the appropriate library manager,
- link these two libraries with the user application.

3. COMPILATION PARAMETERS

The compilation options are replaced by FDM_R4 library configuration options in file "**user_opt.h**".

Before building the two libraries FIPMAN.lib and SEGMENT_FDM.lib, this file must therefore be configured (see COMPILATION PARAMETER chapter of manual ALS 50278).

File mcs_opt.h should not be modified, unless you wish to use the SOFTWARE on a Windows NT PC (see appendix: case of FDM_NT).

Appendix A Example

EXAMPLE OF SEGMENT_FDM SOFTWARE MAKEFILE FOR PSOS

```
#                       MAKEFILE

#

PSS_ROOT= /home/robin/SEGMENT_FDM/psos/pssmcf


include $(PSS_ROOT)/bsps/ut5307/bsp.mk

#*--------------------------------------------------------------------------*

#* List all the files that go into the SEGMENT_FDM here.

#*--------------------------------------------------------------------------*

HEADERS                 = mcs_opt.h mcsprive.h mcspriv2.h mcs.h mcs_os.h

OBJ_DIR = obj/

SRC_OBJ = c.opt \

        $(OBJ_DIR)mcs_user.o \

        $(OBJ_DIR)mcsconfi.o \

        $(OBJ_DIR)mcsheart.o \

        $(OBJ_DIR)mcstools.o \

        $(OBJ_DIR)mcs_os.o

COPTS= -@c.opt
```

```
#*---------------------------------------------------------------------*

#* Macros for running the compiler, assembler, and librarian

#*---------------------------------------------------------------------*

CC= dplus

INC= -I. -I.. -I$(PSS_ROOT)/include -I$(PSS_ROOT)/bsps/devices/fdm

COPTS_FILE1= -c -g -D__DIAB -Xsmall-const=0 -Xsmall-data=0 -DBSP_CACHE_ENABLE

COPTS_FILE2= -t$(CPU)F$(DFP):psos -Xansi -Xstrings-in-text

COPTS_FILE3=  -Xstrict-ansi


LIB=     dar

LIBOPTS= -q

#*---------------------------------------------------------------------*

#* Primary targets:

#*  all:   Build ../libsegmentfdm.a (default)

#*  clean: Clean object directory & intermedate files.

#*  new:   Same as clean with the addition

#*---------------------------------------------------------------------*

all: libsegmentfdm.a

clean:

        rm -f *.opt

        rm -f obj/*.o

new:

        rm -f libsegmentfdm.a

        rm -f *.opt

        rm -f obj/*.o
```

```
#*---------------------------------------------------------------------*

#* Rule to make libsegmentfdm library

#*---------------------------------------------------------------------*

libsegmentfdm.a: c.opt $(SRC_OBJ)

        @rm -f libsegmentfdm.a

        $(LIB) $(LIBOPTS) libsegmentfdm.a $(OBJ_DIR)mcs_user.o

        $(LIB) $(LIBOPTS) libsegmentfdm.a $(OBJ_DIR)mcsconfi.o

        $(LIB) $(LIBOPTS) libsegmentfdm.a $(OBJ_DIR)mcsheart.o

        $(LIB) $(LIBOPTS) libsegmentfdm.a $(OBJ_DIR)mcstools.o

        $(LIB) $(LIBOPTS) libsegmentfdm.a $(OBJ_DIR)mcs_os.o


        @rm -f *.opt

#*---------------------------------------------------------------------*

#* Rules to make files from this directory

#*---------------------------------------------------------------------*

$(OBJ_DIR)mcs_user.o: mcs_user.c  \

        $(HEADERS) \

        makefile

        $(CC) $(COPTS) -o $(OBJ_DIR)mcs_user.o  mcs_user.c


$(OBJ_DIR)mcsconfi.o: mcsconfi.c  \

        $(HEADERS) \

        makefile

        $(CC) $(COPTS) -o $(OBJ_DIR)mcsconfi.o  mcsconfi.c
```

```
$(OBJ_DIR)mcsheart.o: mcsheart.c  \
        $(HEADERS) \
        makefile
        $(CC) $(COPTS) -o $(OBJ_DIR)mcsheart.o  mcsheart.c

$(OBJ_DIR)mcstools.o: mcstools.c  \
        $(HEADERS) \
        makefile
        $(CC) $(COPTS) -o $(OBJ_DIR)mcstools.o  mcstools.c

$(OBJ_DIR)mcs_os.o: mcs_os.c  \
        $(HEADERS) \
        makefile
        $(CC) $(COPTS) -o $(OBJ_DIR)mcs_os.o  mcs_os.c

#*--------------------------------------------------------------------*
#* Utility targets
#*  c.opt - Build the C Compiler options file from macros
#*--------------------------------------------------------------------*
c.opt: makefile
        @echo $(COPTS_FILE1) >  c.opt
        @echo $(COPTS_FILE2) >> c.opt
        @echo $(COPTS_FILE3) >> c.opt
        @echo $(INC)         >> c.opt
```

```
dirs:

        pmkdir $(OBJ_DIR)
```

A_DATA Data transfer service using the MCS protocol, based on a connection established beforehand. This service allows reliable transfer of data units of a size which can exceed the size of a link message (segmentation/assembly, repetition/anti-duplication mechanisms).

A_UNIDATA Data transfer service using the MCS protocol, not based on any connection. This service allows unreliable transfer of data units of a size not exceeding the link message size. These transfers can be acknowledged if the user so wishes.

DLL Frame Protocol data which can be carried in a single FIP message.

FDM_R4 FIP DEVICE MANAGER R4 Software

Frame Data unit sent over the physical layer.

LSAP Link Service Access Point. Data link access point enabling WorldFIP messages to be sent and/or received. The LSAP can be compared to the "FDM message context" implemented in FIP DEVICE MANAGER.

MCS Messaging Common Services. The Software implements these services.

MCS Channel The MCS channel is the operating context allowing MCS messages to be sent and/or received.
In the light version, the configured channel carries out the transfers using the A_DATA protocol.

MFHD MICROFIP HANDLER software. This software can only be used for the MICROFIP component and in particular implements the services of the WorldFIP message system data link layer.

PDU Protocol Data Unit.

Glossary